WEST Search History

Hide Items | Restore | Clear | Cancel

DATE: Monday, June 21, 2004

Hide?	Set Name	Query	Hit Count
		DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=ADJ	
☐	L25	L22 and l9	15
☐	L24	L22 and l3	0
☐	L23	L22 and l4	1
☐	L22	19980508	731
☐	L21	(serial adj4 stream) same (parallel adj4 stream)	1272
☐	L20	L19 NOT l14	10
☐	L19	L18 NOT l15	10
☐	L18	19980508	10
☐	L17	L16 and l9	19
☐	L16	(transmit adj3 processor) near8 (receive adj3 processor)	499
☐	L15	19980508	10
☐	L14	L13 and (instruction adj4 memory)	21
☐	L13	l9 and (aggregate or aggregating or aggregator)	137
☐	L12	L11 and (writeable near8 memory)	1
☐	L11	L10 and (controller)	158
☐	L10	L9 and (instruction adj4 memory)	211
☐	L9	data adj3 stream adj3 processor	1561
☐	L8	(packet adj3 switch) near8 complex near5 function	1
☐	L7	(packet adj3 switch) near8 compled near5 function	0
☐	L6	19980508	2
☐	L5	19980508	70
☐	L4	(packet adj3 switch) same (integrated adj3 circuit)	200
☐	L3	(packet adj3 switch) same (programmable near5 processor)	12
☐	L2	RxSDP and TxSDP	0
☐	L1	DCPIC	1

END OF SEARCH HISTORY

WEST Search History

Hide Items | Restore | Clear | Cancel

DATE: Monday, June 21, 2004

Hide?	Set Name	Query	Hit Count
		DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=ADJ	
☐	L8	L7 and l5	1
☐	L7	L6 and pin	17550
☐	L6	configuration near5 (circuit or circuitry)	97847
☐	L5	19980508	4
☐	L4	L3 and l2	12
☐	L3	interconnection near8 (configurable or configured or configurator)	2167
☐	L2	stream adj4 processor	4896
☐	L1	stream adj4 procerssor	0

END OF SEARCH HISTORY

WEST Search History

Hide Items | Restore | Clear | Cancel

DATE: Monday, June 21, 2004

Hide?	Set Name	Query	Hit Count
		DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=ADJ	
☐	L3	19980508	7
☐	L2	((integrated adj3 circuit) or IC) near8 (receiving adj3 stream adj3 processor) near8 (transmitting adj3 stream adj3 processor)	0
☐	L1	((integrated adj3 circuit) or IC) near8 (stream adj3 processor)	17

END OF SEARCH HISTORY

Document 1 of 1 File: USPT Dec 19, 2000

DOCUMENT-IDENTIFIER: US 6163539 A

TITLE: Firmware controlled transmit datapath for high-speed packet switches

Brief Summary Text (7):

Thus, the high packet transmission rates of modern packet switches virtually mandate the need for dedicated hardware to perform the various complex functions required in the transmit path of Ethernet switching equipment. When these functions are implemented in pure hardware, however, the advantages of software (namely, flexibility and low cost coupled with the ability to realize very complex processing) are lost. The resulting implementation is expensive, difficult to extend and modify, and complex to design.

First Hit Fwd Refs

Generate Collection

L6: Entry 1 of 2 File: USPT Sep 18, 2001

DOCUMENT-IDENTIFIER: US 6292483 B1
TITLE: Apparatus and method for generating an index key for a network switch routing table using a programmable hash function

Abstract Text (1):
A network switch configured for switching data packets across multiple switch ports uses programmable hash functions to generate a hash key for each network address to access an address table storing switching logic. The address table is configured to include a programmable number of bin entries, where each bin entry is configured to reference a plurality of address table entries storing the switching logic information for respective network addresses. The address of an incoming data packet is used to generate a hash key that references a selected one of the bin entries. The switching logic for the corresponding address is then obtained by accessing the appropriate table entry referenced by the selected bin entry. If the number of table entries for a given bin exceeds a prescribed threshold, an external host reprograms the network switch to use another hash key to maintain an efficient access throughput of the address table. Use of programmable hash keys also enables the host processor to use different hash key polynomials for different network configurations to optimize the table access throughput.

Application Filing Date (1):
19971218

Generate Collection

L6: Entry 2 of 2 File: USPT Mar 26, 1985

DOCUMENT-IDENTIFIER: US 4507782 A
TITLE: Method and apparatus of packet switching

Application Filing Date (1):
19820728

Brief Summary Text (6):
The central control unit 34 of the packet switch is usually formed of a programmable processor. The central control unit 34 requires a communication control (CC) 35 to send the packet to a packet switch 13. The communication control 35 sends the packet to a communication control (CC) 45, after having added a new FCS to the packet. The value of FCS added to a packet is usually varied each time the packet is transmitted between terminal and node and between nodes.

First Hit Fwd Refs

Search Forms

Search Results

Generate Collection

Help

User Searches

Preferences

Logout

Search: 1 of 70 File: USPT Nov 4, 2003

DOCUMENT-IDENTIFIER: US 6643285 B1

TITLE: Message based packet switch based on a common, generic bus medium for transport

Application Filing Date (1):
19980217

CLAIMS:

21. The packet switch of claim 8 wherein the components are implemented on application specific integrated circuits (ASICs).

L15: Entry 9 of 10 File: USPT Aug 11, 1998

DOCUMENT-IDENTIFIER: US 5794060 A
TITLE: General purpose, multiple precision parallel operation, programmable media processor

Application Filing Date (1):
19961122

Brief Summary Text (11):
The disadvantages of the prior ASIC approach can be over come by a single unified media processor. The cost advantages of such a unified processor can be achieved by gathering all the many ASIC functions of a broadband media product into a single integrated circuit. Cost reduction is further increased by reducing the total memory area of such a circuit by replacing the multiplicity of small ASIC memories with a single memory hierarchy large enough to accommodate the sum total of all the working sets, and wide enough to supply the aggregate bandwidth needs of all the logic blocks. Additionally, the logic block interconnect circuitry to this memory hierarchy may be streamlined by providing a generally programmable switching fabric. Many of the logic blocks themselves can also replaced with a single multi-precision arithmetic unit, which can be internally partitioned under software control to perform addition, multiplication, division, and other integer and floating point arithmetic operations on symbol streams of varying widths, while sustaining the full data throughput of the memory hierarchy. The residue of logic blocks that perform operations that are neither arithmetic or permutation group oriented can be replaced with an extended math unit that supports additional arithmetic operations such as finite field, ring, and table lookup, while also sustaining the full data throughput of the memory hierarchy.

Brief Summary Text (31):
In one aspect of the invention, the unified stream of media data is processed by storing the stream of unified media data in a general register file. Multi-precision arithmetic operations can then be performed on the stored stream of unified media data based on programmed instructions, where the multi-precision arithmetic operations include Boolean, integer and floating point mathematical operations. The component fields of unified media data can then be manipulated based on programmed instructions that implement copying, shifting and re-sizing operations. Multi-precision mathematical operations can also be performed on the stored stream of unified media data based on programmed instructions, where the mathematical operations including finite group, finite field, finite ring and table look-up operations. Instruction and data pre-fetching are included to fill instruction and data pipelines, and memory management operations can be performed to retrieve instructions and data from external memory. The instructions and data are preferably stored in instruction and data cache/buffers, in which buffer storage in the instruction and data cache/buffers is dynamically allocated to ensure real-time execution.

Detailed Description Text (4):
In the preferred embodiment of the invention shown in FIG. 1, media data streams are communicated to the media processor 12 from several sources. Ideally, unified media data streams are received and transmitted by the general purpose media processor 12 over a fiber optic cable network 14. As will be described in more

detail below, although a fiber optic cable network is preferred, the presently existing communications network in the United States consists of a combination of fiber optic cable, coaxial cable and other transmission media. Consequently, the general purpose media processor 12 can also receive and transmit media data streams over coaxial cable 14 and traditional twisted pair wire connections 16. The specific communications protocol employed over the twisted pair 16, whether POTS, ISDN or ADSL, is not essential; all protocols are supported by the broad band microcomputer 10. The details of these protocols are generally known to those skilled in the art and no further discussion is therefore needed or provided herein.

Detailed Description Text (17):
The current specialized processor approach to media processing is illustrated in the block diagram shown in FIG. 5. As shown in FIG. 5, special purpose processors are coupled to a back plane 70, which is capable of transmitting instructions and data at the upper kilobits to lower gigabits per second range. In a typical configuration, an audio processor 76, video processor 78, graphics processor 80 and network processor 82 are all coupled to the back plane 70. Each of the audio, video, graphics and network processors 76-82 typically employ their own private or dedicated memories 84, which are only accessible to the specific processor and not accessible over the back plane 70. As described above, however, unless video data streams are constantly being processed, for example, the video processor 78 will sit idle for periods of time. The computing power of the dedicated video processor 78 is thus only available to handle video data streams and is not available to handle other media data streams that are directed to other dedicated processors. This, of course, is an inefficient use of the video processor 78 particularly in view of the overall processing capability of this multi-processor system.

Detailed Description Text (31):
The instructions provided in the instruction set for the general purpose media processor 12 control the transfer, processing and manipulation of data streams between the register file 110 and the execution unit 100. The presently preferred width of the instruction path 112 is 32-bits wide, organized as four eight-bit bytes ("quadlets"). Those skilled in the art will appreciate, however, that the instruction path 112 can take on any width without departing from the spirit and scope of the invention. Preferably, each instruction within the instruction set is stored or organized in memory on four-byte boundaries. The presently preferred format for instructions is shown in FIG. 9(a).

Detailed Description Text (35):
The instruction set preferably includes load and store instructions that move data between memory and the register file 110, branch instructions to compare the content of registers and transfer control, and arithmetic operations to perform computations on the contents of registers. Swap instructions provide multi-thread and multi-processor synchronization. These operations are preferably indivisible and include such instructions as add-and-swap, compare-and-swap, and multiplex-and-swap instructions. The fixed-point compare-and-branch instructions within the instruction set shown in Table I provide the necessary arithmetic tests for equality and inequality of signed and unsigned fixed-point values. The branch through gateway instruction provides a secure means to access code at a higher privileged level in a form similar to a high level language procedure call generally known in the art.

Detailed Description Text (52):
Using standard DRAM components, the external memory units 158 achieve bandwidths of approximately two gigabits/second with the standard memories 154. When four such external memory units 158 are coupled via the communication channel 156, therefore, the total bandwidth of the external main memory system increases to one gigabyte/second. As discussed further below, in implementations with two or eight communication channels 156, the aggregate bandwidth increases to two and eight

gigabytes/second, respectively.

Detailed Description Text (94):
To achieve such high rates of data throughput, the media processor includes an execution unit, high bandwidth interface, memory management unit, and pipelined instruction and data paths. The high bandwidth interface includes a mechanism for transmitting media data streams to and from the media processor at rates at or above the gigahertz frequency range. The media data stream can consist of transmission, presentation and storage type data transmitted alone or in a unified manner. Examples of such data types include audio, video, radio, network and digital communications. According to the invention, the media processor is dynamically partitionable to process any combination or permutation of these data types in any size.

CLAIMS:

2. The method defined in claim 1, further comprising the steps of:

pre-fetching instructions and data to fill instruction and data pipelines;

performing memory management operations to retrieve instructions and data from external memory;

storing instructions and data in instruction and data cache/buffers; and

dynamically allocating buffer storage in the instruction and data cache/buffers to ensure real-time execution.

Generate Collection

L23: Entry 1 of 1 File: USPT Sep 25, 2001

DOCUMENT-IDENTIFIER: US 6295299 B1
TITLE: Data path architecture for a LAN switch

Application Filing Date (1):
19980130

Detailed Description Text (12):
The receive buffers operate as serial-to-parallel bit stream converters, and burst x bits of data in parallel to packet memory. The x bits contain n-bit slices of a data packet received from a MAC connected to the corresponding subpath. The data streams from each subpath are written to memory under the control of packet queue manager (PQM) 160. PQM 160 generates the memory address locations at which the slices of data received from the MACs are stored in packet memory 130, in particular, at memory banks 130a, 130b, 130c and 130n. A selector 215, 225, 235 and 245 in respective subpath controllers 120a, 120b, 120c and 120n simultaneously selects receive buffers corresponding to subpaths of the same path and forwards the slices of data held therein to packet memory 130 over data bus 121.

CLAIMS:

2. The packet switch of claim 1, wherein the number of data path controllers is implemented in at least two separate integrated circuits.

4. The packet switch of claim 3, wherein the eight data path controllers are implemented in two integrated circuits, four data path controllers to each integrated circuit.

5. The packet switch of claim 3, wherein the eight data path controllers are implemented in four integrated circuits, two data path controllers to each integrated circuit.

6. The packet switch of claim 3, wherein the eight data path controllers are implemented in eight integrated circuits, one data path controller to each integrated circuit.

First Hit Fwd Refs

Generate Collection

L25: Entry 13 of 15 File: USPT Nov 13, 1990

DOCUMENT-IDENTIFIER: US 4970721 A
** See image for Certificate of Correction **
TITLE: Resource-decoupled architecture for a telecommunications switching system

Application Filing Date (1):
19900116

Detailed Description Text (58):
The TICs, 80 and 81 generate a plurality of serial data streams (e.g. 7) carrying 8 MHz data and these are converted to a plurality of parallel data streams in a pair of host processor circuits (HPC) 82A and 82B and then fed to planes 0 and 1 of the channel switch via a pair of FICs 83A and 83B and fiber links 84A and 84B. As described above in conjunction with FIG. 5, each FIC includes a select circuit for selecting the data from one or the other of the HPCs as well as formatting circuitry to generate DS-512 data and interface to fiber optic links 84A and 84B.

First Hit Fwd Refs

Generate Collection

L5: Entry 1 of 4 File: USPT Oct 27, 1998

DOCUMENT-IDENTIFIER: US 5828858 A
** See image for Certificate of Correction **
TITLE: Worm-hole run-time reconfigurable processor field programmable gate array (FPGA)

Abstract Text (1):
Higher performance is gained through a new architecture which implements a new method of computational resource allocation, utilization and programming based on the concept of Worm-hole Run-Time Reconfiguration (RTR). A stream-driven Worm-hole RTR methodology extends contemporary data-flow paradigms to utilize the dynamic creation of operators and pathways, based upon stream processing in which parcels of data move through custom created pathways and interact with other parcels to achieve the desired computation. These parcels independently allocate the necessary computing resources and data paths as they navigate through the platform. The Worm-hole RTR platform consists of a large number of configurable functional units that perform the custom computations and rich, configurable interconnection pathways between the functional units. Once a computational pathway has been established (sensitized) by the head of the stream parcel, data are processed through the pathway with zero overhead. All ports entering the computing platform serve both to configure operations and pathways and to pass computational data streams. As a result, programming and configuration is not limited to a single port. Configuration through multiple independent ports allows greater concurrency, faster reconfiguration, and fewer computational dependencies, all with relatively low cost in silicon.

Application Filing Date (1):
19960916

Brief Summary Text (10):
According to the invention, higher performance is gained through a new architecture which implements a new method of computational resource allocation, utilization and programming based on the concept of Worm-hole Run-Time Reconfiguration (RTR). The concept is derived from a known method of routing data through a network known as "worm-hole" routing in which routing involves a path through the network from the source to the destination that is maintained until the entire data set has been delivered. The invention enhances this concept with a stream-driven Worm-hole RTR methodology which extends contemporary data-flow paradigms to utilize the dynamic creation of operators and pathways. Worm-hole RTR is based upon stream processing in which parcels of data move through custom created pathways and interact with other parcels to achieve the desired computation. Much like worms tunneling through an apple, these parcels independently allocate the necessary computing resources and data paths as they navigate through the platform. The Worm-hole RTR configurable platform consists of a large number of configurable functional units that perform the custom computations and rich, configurable interconnection pathways between the functional units. Once a computational pathway has been established (sensitized) by the head of the stream parcel, data are processed through the pathway with zero overhead.

Detailed Description Text (3):
Worm-hole RTR addresses the problems of current FPGAs and, in future applications,

provides a run-time reconfiguration method for large scale systems. It is useful as a method for rapidly programming data flow graphs into large systems using a distributed control scheme. However, Worm-hole RTR is not limited to data flow computation and can be adapted to work with other computing environments. The essence of the Worm-hole RTR concept is formed from independent self-steering streams of programming information and operand data that interact within the architecture to perform the computational problem at hand. The method of computation itself can be compared to that of "pipenets", as discussed by K. Hwang et al. in Advanced Computer Architecture, McGraw-Hill (1993), at pages 442-446, in which multiple intersecting pipelines (streams) are formed within the processor to perform a task. One application of Worm-hole RTR is as a high speed configuration methodology for such a system.

CLAIMS:

12. A data stream driven Worm-hole run-time reconfigurable field programmable gate array (FPGA), comprising:

a plurality of independent bi-directional data ports for receiving one or more data streams and outputting one or more data streams, each said data stream containing variable length self-configuring header information and variable length data;

a plurality of interconnected and configurable functional units capable of performing programmed operations as defined by said header information;

at least one interconnection network responsive to said header information for configuring pathways interconnecting data ports and configurable functional units; and

at least one stream controller connected to said interconnection network, said field programmable gate array being dynamically configurable by said stream controller at run-time through said plurality of independent bi-directional ports to partially reconfigure for one task while simultaneously performing computations for another task.

First Hit Fwd Refs

Generate Collection

L5: Entry 2 of 4 File: USPT Jul 21, 1998

DOCUMENT-IDENTIFIER: US 5784636 A
TITLE: Reconfigurable computer architecture for use in signal processing applications

Application Filing Date (1):
19960528

Brief Summary Text (5):
SRAM (static random access memory)-based FPGAs consist of logic and interconnection resources that are configured by program data stored in internal SRAM cells. The same FPGA components can be reconfigured an unlimited number of times, thereby allowing the same component to implement many different functions.

Detailed Description Text (17):
The compiler implements the remainder of the application using code sequences for the scalar processor. The program fragments include the code for the reconfiguration of the configurable logic resource. The instruction stream of the scalar processor is also responsible for the high level task sequencing using the Toggle Bus transactions and associated pipeline control mechanisms. At the lower levels of the computation, coordination of the operations between the functional units is managed by the "autonomous pipeline" capabilities of the configurable logic array which will be described later. Finally, the scalar processor performs all of the remaining processing tasks which have not been assigned to the configurable logic or parallel processing modules. An aspect of the behavior of RSP based systems is that the same subroutine may be executed in three different ways: (1) totally in the configurable logic if this resource is not otherwise in use at the specific point in the computation, (2) partially in the configurable logic if only a portion of the configurable logic resource is available or (3) totally in the scalar processor when all of the configurable logic resource is employed performing computations which result in better performance.

L5: Entry 3 of 4 File: USPT Oct 28, 1997

DOCUMENT-IDENTIFIER: US 5682491 A
TITLE: Selective processing and routing of results among processors controlled by decoding instructions using mask value derived from instruction tag and processor identifier

Application Filing Date (1):
19941229

Detailed Description Text (36):
Turning now to FIG. 7, still another interconnection arrangement can be configured using the interconnection switches 106 and the respective processing elements 100 in the array 50D to form a linear ring organization shown in FIG. 7. In FIG. 7, the processing elements 100, 100A and 100B in the left hand column of the array 50D, can cascade their results so that processor 100 outputs its results over line 101 to processor 100A, processor 100A outputs its results over line 101A to processor 100B (for N=3), and then processor 100B outputs its results over line 103 to processor 100. This linear ring organization is configured by virtue of the instruction decode 118 and each respective processing elements 100, 100A and 100B, appropriately combining the tag value 300 from the common instruction broadcast to all of the processors 100, 100A and 100B, so that an appropriate interpretation is made of the opcode portion of the instruction, to interconnect the interconnection switch 106, into the cascaded configuration shown for the processors in FIG. 7. In this manner, the inter-processor configuration for the processors in an array of processors, can be dynamically reconfigured. Examination of FIG. 7 will show that a similar configuration of a linear ring is formed for the processors 100', 100'A and 100'B for the instruction stream output by the sequencer processor 206'. Similarly, a linear ring organization is created for the processors 100", 100"A and 100"B executing the instructions output from the sequencer processor 206".

Detailed Description Text (38):
Reference can now be made to FIG. 9 which shows a diagonal fold 4.times.4 nearest neighbor mesh array 50F. In the processor organization shown in FIG. 9, each of the processors, for example the processor 100'C, is interconnected to its four nearest neighbor processors by means of links labelled north, east, west and south, which are the links 122 previously referred to as being selectively configured by the interconnection switch 106 for the processor 100'C. The interconnection of the links 122 is accomplished by control signals output from the instruction decode 118 after appropriate interconnection of an instruction by combining the tag value 300 with a processor identity, to uniquely interpret the instruction and provide interconnection signals for the interconnection switch 106. The links 122 enable the four nearest neighbors for the processor 100C' to be connected for an exchange of the arithmetic results from the processor's arithmetic operations in the arithmetic elements 120. Also shown in FIG. 9 is the co-location of two processors at a single node 902, 904, or 906, etc., respectively. Each pair of processors at a node, for example at node 902, can communicate its arithmetic results to its nearest neighbors in the manner previously described. In addition, in an alternate embodiment of the invention, each processor pair, for example 100A and 100B in the node 902 of FIG. 9, can communicate its results to the other processor in the node, using the interconnection switch 106, as described above.

First Hit Fwd Refs

Generate Collection

L5: Entry 4 of 4 File: USPT Jun 6, 1989

DOCUMENT-IDENTIFIER: US 4837676 A
TITLE: MIMD instruction flow computer architecture

Application Filing Date (1):
19880815

Brief Summary Text (9):
Still other attempts to improve processing speeds and to minimize contention problems are described in "Parallel Processor Systems, Technologies, and Applications," Symposium, June 25-27, 1969, Chapter 13 entitled "A Multiple Instruction Stream Processor with Shared Resources," M. J. Flynn, A. Podvin and K. Shimizu. Here, a parallel computer system organization is described using individual computers, each of which contains its own data and control registers but lacks the more substantial execution facilities which, in turn, are shared by all machines. Sharing is accomplished by closely synchronized time-phased switching. Heavy pipelining of the execution resources is used in an effort to provide maximum operational bandwidths. The pipelining factor for each of the execution functions of the execution resources is necessarily closely related to the synchronizing factor of the individual computers. The system based upon this organizational concept is claimed to avoid many of the contention problems associated with shared resource systems.

Drawing Description Text (3):
FIG. 2 depicts an instruction flow computer according to this invention as configured employing a multi-stage interconnection network;

Other Reference Publication (17):
Chapter 13, a Multiple Instruction Stream Processor with Shared Resources, M. J. Flynn et al.; Parallel Processor Systems, Technologies, and Applications, Symposium, Jun. 25-27, 1969; pp. 251-286.

L15: Entry 4 of 10 File: USPT Apr 25, 2000

DOCUMENT-IDENTIFIER: US 6055619 A
TITLE: Circuits, system, and methods for processing multiple data streams

Application Filing Date (1):
19970207

Brief Summary Text (9):
According to an embodiment of the principles of the present invention, an audio information processing subsystem is provided which includes a stream processor for simultaneously processing multiple streams of audio data. A program memory is coupled to the stream processor by a first bus for storing instructions for controlling the processing system. A data memory is coupled to the stream processor by a second bus. Direct memory access circuitry is included for controlling direct memory accesses to a selected one of the program and data memories.

Detailed Description Text (11):
Accelerator 200 is based upon a Harvard architecture having a program memory 202 and associated program bus 111 separate from the data memory and its associated data bus. Preferably, program memory 202 is organized as 3.5K words.times.40 bits per word. The data memory is arranged in two address spaces, with each providing operands to stream processor (core). One address space includes parameter RAM 203 along with coefficient ROM 201, previously discussed. Parameter RAM 203 preferably has a 3.2-K word.times.32-bit per word organization. Parameter RAM 203 contains, among other things, stream parameters for audio sample streams, wavetable parameters for voice, 3-D sound parameters and 3-D sound head related transfer function (HRTF) filtering parameters. A second address space includes sample RAM 105 which receives and manages data over DMA bus 211 and PCI interface 207, and also receives data to and from stream processor 100 across bus 213.

Detailed Description Text (12):
Each of these memories 203/204 possesses two interfaces, one for stream processor 100, and the other for DMA engine 208. DMA 208 is responsible for providing instructions and data to stream processor 100 via the DMA memory interface, and for transferring results from these memories back out to the host. Each of the functional memories 202/203/204 is structurally composed of two physical memories, with all data and instructions to odd addresses residing in one memory and all data and instructions to even addresses residing in the other memory. of the two memory interfaces included with each physical memory, the SP interface is the higher priority interface, with the DMA interface being of lower priority. It should be noted that each of these memories is not truly two-ported, rather, they are single port RAMs having two control and data interfaces.

Detailed Description Text (25):
The scatter/gather feature according to the principles of the present invention, at least in part, is directed to the problems of transferring bursts of data into the sample memory buffers. In particular, the instantaneous bandwidth can be increased by pushing down the priority of the page table reads. By initiating the next request for access early, the next page table read can be performed at anytime during the current page accesses, the page table read priority reduced, and the bus freed to transfer bursts of audio stream data. A PCI bus, although it has an

aggregate, best case bandwidth of 133 Mbytes per second, may make available bandwidth to a single device on the bus as little as 10 Mbytes per second. Among other things, a given device on the bus may have to compete with 6 or more other devices on the bus which also require bandwidth.

Detailed Description Text (180):
FIG. 26A is a conceptual block diagram of the primary serial input and output ports. The primary output port includes a set of first in/first out registers 2600 for pipelining data received from stream processor 100. The data pipelined from output sample FIFOs 2600 are provided to a data shifter/formatter 2601 under the control of command addresses and data presented through port timing and control 2606. Serial data is then output from shifter/formatter 2601 through the serial output port (SDOUT) 2602.

Detailed Description Text (263):
A preferred procedure for stopping stream processor operation at the frame boundaries for debugged purposes is also provided. In this case, the STPFR bit in SPCR (FIG. 56) register is set. The STFR bit is monitored in the SPCR register (FIG. 60) for one to zero transitions. Stream processor 100 is now stopped and emulated frame mode is automatically switched on. The STPFR bit in the SPCR register is cleared by hardware when the SP stops at the next frame. Then the SDSR byte is monitored until the current stream equals zero (to make sure all pending DMAs are complete). Next, the DRQEN bit in the SPCR register is cleared to prohibit new DMA request generation. Finally, to verify that the stream processor 100 has stopped at the frame boundary and not in response to a deep sleep instruction, the program memory is examined to assure that deep sleep did not cause the STOP condition.

Detailed Description Paragraph Table (118):
APPENDIX 4

mode
({accOnly .vertline. auxOnly .vertline. acc:aux .vertline. signedSat .vertline. unsignedSat}, . . . Change the MAC unit, ALU saturation, Y-register/sample routing, parameter/coefficient routing, and/or parameter/coefficient data size modes
Operation: mode({accOnly .vertline. auxOnly .vertline. acc:aux .vertline. signedSat .vertline. unsignedSat}, . . . {ySingle .vertline. yDual}, {,, .vertline. s[hl]:[hl], p[hl]:[hl], p(16 .vertline. 20)}); Where: accOnly specifies to use MAC-unit 0 only, auxOnly specifies to use MAC-unit 1 only, acc:aux specifies to use both MAC units concurrently, signedSat specifies to use signed ALU saturation mode, unsignedSat specifies to use unsigned ALU saturation mode, ySingle specifies that when the Y input to the multiplier is a register-file register, it is to be interpreted as broadcast to both MAC units, yDual specifies that when the Y input to the multiplier is a register-file register, it is to be interpreted as going to MAC-unit 0. The source for MAC-unit 1 is the paired register of that specified (i.e. r4 implies r4 goes to MAC-unit 0 and r5 to MAC-unit 1) sh:h, sh:l, specifies that when the sample AGU sources the Y input to the sl:h, sl:l multiplier ("rsa0-3++ . . . or rsd0-3), the first selection (h or l before `:`) specifies the routing into MAC-unit 0 from the sample data and the second selection (h or l after `:`) specifies the routing into MAC-unit 1. For single MAC-unit mode, you should specify "sh:h" since when single samples are read from sample memory, they are placed into the most-significant 16-bits of the sp.sub.-- yd. ph:h, ph:l, specifies that when the parameter/coefficient AGU sources the X pl:h, pl:l input to the multiplier (*r0-3{++ . . .}, *(r0-3+1)), the first selection (h or l before `:`) specifies the routing into MAC-unit 0 from the parameter data and the second selection (h or l after `:`) specifies the routing into MAC-unit 1. For 20-bit data, "ph:h" should be specified since 20-bit data consumes the entire 32-bit word and is aligned in the upper 20 bits of the word. p16, p20 specifies the bit-width of data from the parameter/coefficient AGU into the X input of the multiplier. 20-bit data is broadcast to both MAC units (ph:h only). For 16-bit data, any of the above parame-

ter/coefficient routings are valid. Restrictions: None Description: Mode-change instruction. This instruction can change the MAC-unit, ALU saturation, sp.sub.-- vd register-file source, or sample/parameter dual data routing modes. It also may specify the bit-width of data from the parameter/coefficient AGU. The MAC unit and routing modes allow single-instruction, dual-data control of the dual MAC units. All mode bits except the ALU saturation mode may be set bu modifying the MACmode register. See page 14 for a description of this register. The ALU saturation mode may be set by modifying the status register (see page 17). Both of these register may be modified using the extended register move instructions on page 124. Flags affected: None Example: mode (acc:aux,, sh:l, ph:h, p20); // Set dual MAC-unit mode, route high/low words of sample data to MAC units 0/1, and select 20-bit param. AGU data mode(accOnly,,,,); // Set single MAC-unit mode Opcodes:

ALUop:

aluOperands aluOpcode 18 17 16 15 14 13 12 11 10 [9:7] [6:4] [3:0] samp param pa sprs 0 yr yr MuSatM 000 0000 mode({MuSatM}, {yr32}, (,,lsampleAGU route route sz upd upd 32 route, paramAGU route, param, size}) 0 0 0 000 NOP (ALUop)

Detailed instruction operation: // Update MAC-unit or saturation mode as appropriate if (aluOperands[9] == `0`) { // Insure that "000" is a NOP, // MACunitEnab[0] indicates whether MAC-unit 0 is active, and // MACunitEnab[1] indicates whether MAC-unit 1 is active. if (aluOperands[8:7] != "00") MACunitEnab[1:0] .rarw. aluOperands[8:7] // A change to the parallel MAC-unit mode is staged over 3 clock cycles. In the // execution cycle of the current instruction, the multiplier is switched to the // new mode. Two clock cycles later, the 39-bit adder is switched into the // specified mode. This staging allows for the normal flow of MACs through // the unit to be cleanly changed from single to dual and back. } else // aluOperands[9] ==`1` { // Insure that "11x" is reserved for future use, // satMode indicates signed saturation when 0 and unsigned when 1 if (aluOperands[8] == `0`) satMode .rarw. aluOperands[7] // A change in the saturation mode affects all 16-bit ALU adds, subtracts, and // shifts that specify to use saturation (operator preceded by $). } // Update mode for register-file sources to the Y input of the MAC units if(aluOperands[11] ==`1`) yr32 .rarw. aluOperands[10] // Update sample routing parameter/coef. routing, and parameter/coef. size if(aluOperands[13] == `1`) { sAGUroute[1:0] .rarw. aluOperands[18:17] pAGUroute[1:0] .rarw. aluOperands [16:15] pAGUsize20b .rarw. aluOperands[14] } mode({accOnly .vertline. auxOnly .vertline. acc:aux .vertline. signedSat .vertline. unsignedSat}, . . . pr = +/- srcx * srcy, after a 1 cycle delay Load the specified srcx and srcy operands into X and Y for multiplication with an optional sign change of the resulting product

---

Operation: pr = +/- srcx * srcy, after a 1 cycle delay where: srcx is r8, r9, rA, . . ., rF, hi20(r54, r76, r98, . . ., rFE), r5:r4, r7:r6, . . ., rF:rE, acc0-3$, *r0-3, *(r0-3+1), *r0-3++1, *r0-3++rci0, *r0-3++rci1, rsa0frac, 1.0-rsa0frac, c-1.0, abs(srcy), square(srcy) srcy is r4, r5, r6, . . ., rF, rsdN, (rsdN = *rsaN++1), (rsdN = *rsaN++rsiM) where N = 0-3 and M = 0-2 Restrictions: If srcx specifies a register-file register source (r8, r9, . . ., rF, hi20(r54, r76, . . ., rFE), or r5:r4, r7:r6, . . ., rF:rE), then the accumulator may not be loaded from a register-file regis- ter in a parallel ALUop instruction (both accumulator load and srcx use the sp.sub.-- gx bus for register-file sources). If srcx specifies a saturated accumulator source (acc0$, acc1$, acc2$, or acc3$), then the accumulator may not be loaded from a register-file register in a parallel ALUop instruc- tion (both accumulator load and srcx use the sp.sub.-- gx bus for register-file sources). A paral- lel accumulator shift operation is also prohibited since both instructions use the sp.sub.-- ad (MAC unit result) bus. If srcx specifies a parameter/coefficient memory source (*r0-3, *(r0-3+1), *r0-3++1, *r0-3++rci0/1), then a parameter/coefficient memory load may not be specified in a par- allel ALUop instruction. A parameter/coefficient memory store may not be specified in either a parallel ALUop instruction nor in the immediately preceding instruction. Also note that the address register (r0-3) and any increment register used (rci0/1) must be loaded at least 2 cycles before being used in the parameter/coefficient AGU. See the pipeline illustration in the description of "Rldst{16} = {lo .vertline. hi} *bs

(++postInc)" on page 119. If srcy specifies a register-file register, then a parameter memory store may not be speci- fied in a parallel ALUop instruction (parameter store and srcy share a register-file port). If srcy specifies a sample memory load (. . . *rsaN . . .), then the accumulator may not be loaded from a sample AGU source in a parallel ALUop instruction. A sample store may not be specified in either a parallel ALUop instruction nor in the immediately preceding instruction. Also note that the sample address register (rsa0-3) and any sample incre- ment register used (rsi0-2) must be loaded at least 3 cycles before being used in the sam- ple AGU. See the pipeline illustration in the description of "AccDest = sampleAGUsrc" on page 111. If srcy specifies a sample data register load (rsdN), then the accumulator may not be loaded from a sample AGU source in a parallel ALUop instruction (the two would col- lide upon reading rsdN over the Y bus). Note that the latency of the multiplier is two cycles (the multiply instruction loads X and Y during its execution cycle and the multiply automatically takes place in the subsequent

CLAIMS:

13. An audio information processing subsystem fabricated on a single-integrated circuit chip comprising:

a stream processor for simultaneously processing multiple streams of audio data;

a program memory coupled to said stream processor by a first bus for storing instructions for controlling said processing subsystem;

data memory coupled to said stream processor by a second bus; and

direct memory access circuitry for controlling direct memory accesses to a selected one of said program and data memories, said direct memory access circuitry defining a plurality of buffers in said data memory, each buffer associated with one of said data streams.

17. The audio information processing subsystem of claim 13 and further comprising trapping circuitry for monitoring instructions being transferred from said program memory to said stream processor and initiating a program branch in response.

29. A stream processing system comprising:

a PCI bus;

a host processing subsystem coupled to said PCI bus including:

a central processing unit;

a host memory; and

circuitry for interfacing said central processing unit and said host memory with said PCI bus; and

a stream processing subsystem coupled to said PCI bus including:

a stream processor for processing a plurality of streams of data simultaneously received, said stream processor including a direct memory access engine operable to exchange information with said host system using direct memory accesses, said direct memory access circuitry controlling a plurality of buffers in memory, each said buffer associated with a corresponding one of said streams of data; and

circuitry for interfacing said stream processor with a source of multiple streams of data.

First Hit Fwd Refs

Generate Collection

L8: Entry 1 of 1 File: USPT Jul 21, 1998

DOCUMENT-IDENTIFIER: US 5784636 A
TITLE: Reconfigurable computer architecture for use in signal processing applications

Application Filing Date (1):
19960528

Brief Summary Text (5):
SRAM (static random access memory)-based FPGAs consist of logic and interconnection resources that are configured by program data stored in internal SRAM cells. The same FPGA components can be reconfigured an unlimited number of times, thereby allowing the same component to implement many different functions.

Brief Summary Text (7):
A basic reconfigurable system consists of two elements: a reconfigurable circuit resource of sufficient size and complexity, and a library of circuit descriptions (configurations) that can be down-loaded into the resource to configure it. The reconfigurable resource would consist of a uniform array of orthogonal logic elements (general-purpose elements with no fixed functionality) that would be capable of being configured to implement any desired digital function. The configuration library would contain the basic logic and interconnect primitives that could be used to create larger and more complex circuit descriptions. The circuit descriptions in the library could also include more complex structures such as counters, multiplexers, small memories, and even structures such as controllers, large memories and microcontroller cores.

Brief Summary Text (9):
Fixed interfaces are a direct result of connecting multiple chips (FPGAs and non-FPGAs) together to form a larger reconfigurable array. Because these chips are connected together using conventional printed-circuit board (PCB) methodology, the pin interconnections between individual FPGA devices are fixed at the time the system is constructed. Programmable interconnect devices such as crossbars can provide limited flexibility between FPGAs so that pin assignments can be changed to suit the needs of the application. However, programmable interconnects add additional delay to the paths between FPGAs, and the pins between the interconnect device and the FPGA remain fixed.

Brief Summary Text (11):
Although the introduction of fixed interfaces and fixed functions has allowed practical reconfigurable systems to be constructed, the resulting non-uniformity of these systems makes them more difficult to utilize. Design is simplified in a uniform design medium because the only physical constraints that impact the design process are the I/O pins that bring data on and off chip. However, once fixed circuitry is introduced, not only is part of the system's functionality fixed, but some of the original flexibility of the FPGAs is also lost.

Brief Summary Text (12):
The loss of flexibility impacts both implementation and performance. For example, circuit implementations are less efficient with multiple chips because the off-chip interconnect used to communicate between chips is less dense and of lower

performance than what is available on-chip. Circuit modules that must be split across multiple FPGAs will typically exhibit lower performance than if the module fit into a single FPGA. Partitioning a circuit implementation across a set of fixed and unfixed components is also very difficult. For example, if it is determined that a fixed-function memory is necessary for a given application, any circuitry that communicates with the memory will need to be placed in FPGAs with pins that are connected to the multiplier.

Detailed Description Text (17):
The compiler implements the remainder of the application using code sequences for the scalar processor. The program fragments include the code for the reconfiguration of the configurable logic resource. The instruction stream of the scalar processor is also responsible for the high level task sequencing using the Toggle Bus transactions and associated pipeline control mechanisms. At the lower levels of the computation, coordination of the operations between the functional units is managed by the "autonomous pipeline" capabilities of the configurable logic array which will be described later. Finally, the scalar processor performs all of the remaining processing tasks which have not been assigned to the configurable logic or parallel processing modules. An aspect of the behavior of RSP based systems is that the same subroutine may be executed in three different ways: (1) totally in the configurable logic if this resource is not otherwise in use at the specific point in the computation, (2) partially in the configurable logic if only a portion of the configurable logic resource is available or (3) totally in the scalar processor when all of the configurable logic resource is employed performing computations which result in better performance.

Detailed Description Text (33):
ALP Adaptive Logic Processor. A programmable logic structure which allows the implementation of application specific logic circuits for controlling external components and computation pipelines. The ALP consists of an array of logic cells, programmable interfaces to the input/output pins of an RSP device, programmable interconnects between the logic cells, and access to the Pipeline bus Array (PBA) data. The entire ALP may be dedicated to implement one function or several smaller functions may operate concurrently. RPIC configures the ALP circuits using data in the MPM or from the Toggle Bus. The circuits in the ALP are generally changed several times during the execution of a typical program. Circuits in the ALP which are not in the process of configuration change remain active.

Detailed Description Text (50):
Q Secondary Toggle Bus interface bus. When only one RSP is used in a system, the Q bus may be used as a second Tristate bus or as a reconfigurable set of input/output pins for external memory, sensor and actuator connection. When several RSP components are used in a SIMD array, the Q and P lines are connected together using a "shuffle wiring" pattern. P and Q then operate together with the set of TBT circuits to create a MIN parallel processing interconnect network. (see FIG. 17)

Detailed Description Text (54):
In addition to these primary data and address interface busses, several individual control signal pins on an RSP chip are dedicated for clock input, external memory control, and so forth.

Detailed Description Text (65):
The Adaptive Logic Processor (ALP) consists of an array of programmable logic cells and an array of programmable switches which allow data flow between the cells. These elements form a configurable logic resource which is incorporated in the RSP architecture. The ALP includes the Pipeline Bus Array (PBA) and reconfigurable input/output pins. FIG. 5 illustrates the basic elements of ALP 120 in an implementation having 32 horizontal cells and 64 vertical cells.

Detailed Description Text (66):

In the general case, ALP 120 consists of three circuit layers which perform functions specific to the RSP architecture. The base layer is called the "core cell array" and consists of a two dimensional array of core cell 150 logic circuits. Each of these core cell circuits includes a configuration register which may be loaded under control of the RPIC block when a "reconfiguration" instruction is executed. The value in the configuration register defines the logic function to be performed by the core cell and defines connections for the logic input and output signals of the core cell. The configured function of a core cell is typically a multiple input, multiple output Boolean operation which may include one or more data flip flops. In a typical implementation, some or all of the signals needed to implement the Boolean operation include connections to the nearest neighbor core cells. This represents the lowest level of core cell interconnections.

Detailed Description Text (67):
The second logical layer of the configurable logic array (ALP) consists of signal lines and switches which allow transmission of signal values over a distance of several core cells. This is referred to as a "Local Bus Array". Each core cell may receive input from one or more local bus array wiring segments and may provide data to one or more local bus array segments. Each set of local bus wiring paths span an area termed a "core block" 152 in the figure. The circuits which allow signal interchange between core blocks are called "repeater" circuits (not shown in the figure). Each repeater circuit also includes a configuration register which defines how signals flow from one core block to another. The second logical layer of the configurable logic array also includes means to connect signals in the array to the reconfigurable I/O pin drivers 153 which represent the primary interface to an RSP device. Each pin on the RSP has a programmable interface circuit which is controlled by a configuration register. This allows each pin to have a default meaning (such as one of the input lines from the Toggle Bus) or a reconfigured use (logic input, logic output, or bi-directional tristate bus driver). In the figure, reconfigurable I/O pin resources 153 are shown as the "A", "B", "C", "D", "P", and "Q" lines. Detailed control signals are also typically made available at the periphery of the configurable logic array. In the second logical layer, configurable clock and reset signals are distributed throughout the array to implement the pipeline stages of a typical application.

Detailed Description Text (69):
Each core cell 150 of ALP array 120 may be programmed to implement any of a large number of logic functions. These include logic AND, OR, XOR, multiplexer, and flip-flop operations. As noted, the periphery of the core cell array allows connection to the programmable I/O pins 153 on a RSP chip. Each core cell may be programmed (by means of the programmable interconnection switches) to access one or two signals from the four neighboring core cells. Each core cell may also receive data or transmit data on a Local Bus Array (LBA) segment (shown in the figure using dashed lines). As noted, the LBA connections for a subset of the core cells form what was referred to as a core block. Superimposed on the core cell and LBA network is the Pipeline Bus Array (PBA) 132, carrying signals which cross the entire array in the horizontal direction. As noted, circuits called "repeaters" allow the LBA data to be extended to neighboring blocks, the programmable I/O pins and the PBA. A set of programmable clock and reset signal generators 154 traverse the array in the vertical direction. These signals allow each column to operate as a pipeline stage in the array. The PBA data lines 132 allow connection to the RD, WD,RX, WX, RA, and WA busses in the RSP. The PCP control lines 134 represent programmable control and status lines which are connected directly to the RPIC for ALP pipeline control. FIG. 5 illustrates the typical area used by a programmable pipeline segment 160 of a computation. Each pipeline segment consists of a number of pipeline data path stages in the "pipeline data path" area 156 of the ALP and the associated pipeline control circuitry in the "pipeline control area" 158.

Detailed Description Text (80):
As noted, the Reconfigurable Pipeline Instruction Control (RPIC) unit in the RSP is

responsible for controlling program execution, coordinating the activities of data flow between the processing units, and configuration of the programmable circuits in the ALP. The basic functional operations of the RPIC may be divided into five categories:

Detailed Description Text (109):
FIG. 10 shows an example circuit for implementing a ALP core cell. This circuit is the functional equivalent of the CLAy core cell. The circuit logic consists of two sections: (1) a configuration data circuit 202 which defines the cell function and input connections and (2) a function circuit 201 which performs the operation defined by the configuration data.

Detailed Description Text (110):
The configuration logic for a core cell consists of a 16-bit register (DLATCH 203 and FLATCH 204) and a configuration decoder circuit (DC) 206. The output of the configuration register and decoder are used to select the functional signal paths for the function to be performed by the core cell. For example, configuration register bits R0 and R1 are used to select the "A" logic input to the cell from one of the four neighbor cells. The lower eight bits of the configuration register are loaded by first asserting the CSL and SET signals. The row select signal (RS) is then strobed to load the data bits. The upper eight bits are similarly loaded using the CSH signal instead of the CSL signal. The current configuration data value may be read from the CD lines by setting the CSH or CSL signals low. The current value of the core cell flip flop may be read by asserting the FRD signal. This allows the application program to directly read a pipeline register value.

Detailed Description Text (126):
As was discussed with reference to FIG. 5, a contiguous rectangular array of core cells (CC) with the same local bus connections is called a core block. FIG. 12 shows the general plan of the core block structure for a core block of four horizontal columns and four vertical rows. Each core block 220 is connected to neighboring core blocks and to the Pipeline Bus Array using horizontal repeater (HR) 222 and vertical repeater (VR) 224 circuits. The distribution of configuration data is accomplished by the horizontal repeater circuits which read or write the CD data based on the PBA lines that run throughout the array. For a small ALP circuit, the PBA lines may be used directly for the configuration data. FIG. 13 shows the connections which are part of the interface between the core cells and repeater circuits at the corner of four core blocks.

Detailed Description Text (175):
FIG. 19 is a state diagram showing the operational flow for generating a Pipeline Enable signal for the program initiated and autonomous data transfer operations. One skilled in the art can readily produce the circuitry needed to implement the state diagram shown. At power up and after a hard reset, all pipelines are driven to the "Disabled" state. This forces the pipeline enable signals (PENj) LOW which disables the other operations. The pipeline remains disabled as long as a configuration sequence is being executed (signal CFG is HI) and the configuration register active control bit is LOW. When CRAj is HI and no configuration sequence is being executed, the circuit checks the ACTj signal from the ALP in the "Check" state. When ACTj is asserted LOW, the circuit enters the "Enabled" state and generates a HI value on the PENj signal for the pipeline. A pipeline will then be disabled if another configuration sequence is executed or an operation is performed that releases the ACTj signal.

Detailed Description Text (189):
The ALP in an RSP component may receive configuration data from one of three sources: (1) external data (on the RX bus from the Toggle Bus Transceiver) originating in a bus master RSP component, (2) external data from a memory or system bus or (3) from the multiport memory. At power up, an input pin indicates that an RSP is to be configured using external data or that the RSP is to address

an external memory for the configuration data. Configuration of the ALP from the on chip memory provides a fast mechanism for reconfigurable pipeline changes but requires the memory to be first loaded from an external source.

Detailed Description Text (193):
Reconfigurable I/O Pins (RIOPs)--the periphery of the ALP array includes family member dependent access to reconfigurable I/O pins as well as access to RSP internal control signals. This serves to extend the periphery of the array to neighboring arrays using "inter quad" signals In the RSP case, each I/O has access to RSP internal signals as well as the usual reconfigurable pad meaning. Addressing of periphery cells is device dependent by necessity. There are several differences in the behavior of the RIOPs in the ALP.

Detailed Description Text (222):
I/O Pipeline. Interface to external sensors and actuators using the configurable I/O pins (such as the A and B data sets). This reduces the number of external components needed to implement a system. In this role, the ALP is used as a traditional FPGA structure to interact with the control signals of the external devices in a flexible manner. The PDP program then accesses the external data using pipeline bus read and write instructions. The ALP may provide pre- and post-processing functions such as bit assembly and reassembly.

First Hit Fwd Refs

☐ Generate Collection

L3: Entry 1 of 7 File: USPT Jan 9, 2001

DOCUMENT-IDENTIFIER: US 6172990 B1
TITLE: Media access control micro-RISC stream processor and method for implementing the same

Application Filing Date (1):
19971112

Detailed Description Text (25):
In one embodiment, micro-RISC stream processor 114b is a user programmable in-line packet processing engine that is capable of rapidly parsing through received packet data to build user defined data structures that may be appended to the beginning of the received packet before being transferred to multi-packet queue FIFO Rx 108. To initialize a user programming operation of micro-RISC stream processor 114b, the user may configure a software instruction set designating the type of parsing to be performed on in-coming packet data, as well as the type of data structure to build and append to respective packets through the use of a graphical user interface (GUI). Once the instruction set is configured, the software instruction set is compiled to executable "microcode" by the host's CPU, and then transferred into hardware memory locations resident in the integrated circuit core of micro-RISC stream processor 114b.